Code Documentation: Fuel Filter v 2.0

Implementation: <https://github.com/hirushig/fuel-filter>

### **1. Import Libraries**

from pymongo import MongoClient

import pandas as pd

import numpy as np

from scipy.signal import firwin, lfilter, filtfilt

import matplotlib.pyplot as plt

* **MongoClient** from pymongo is used to connect to a MongoDB database.
* **pandas**: Used for data handling and analysis.
* **numpy**: Used for numerical computations.
* **scipy.signal**: Contains signal processing functions, including firwin for filter design and lfilter/filtfilt for filtering.
* **matplotlib.pyplot**: Used for plotting data.

### **2. Define the FIR Low-Pass Filter Function**

def fir\_lowpass\_filter(data, cutoff, fs, numtaps=101):

nyquist = 0.5 \* fs

normal\_cutoff = cutoff / nyquist

fir\_coeff = firwin(numtaps, normal\_cutoff)

filtered\_data = lfilter(fir\_coeff, 1.0, data)

return filtered\_data

* **Purpose**: This function applies a Finite Impulse Response (FIR) low-pass filter to the data, filtering out high-frequency noise that could be caused by road conditions.
* **Parameters**:
  1. data: The signal data (fuel level) to filter.
  2. cutoff: The cutoff frequency for the filter.
  3. fs: Sampling frequency.
  4. numtaps: The number of filter coefficients. Higher values result in a smoother filter but increase computation.
* **Steps**:
  1. **Calculate the Nyquist frequency** as half of the sampling rate.
  2. **Normalize the cutoff frequency** by dividing it by the Nyquist frequency.
  3. **Design the FIR filter** coefficients with firwin.
  4. **Apply the filter** using lfilter, returning the filtered data.

### **3. Define the Moving Average Low-Pass Filter Function**

def moving\_average\_lowpass\_filter(data, window\_size):

return np.convolve(data, np.ones(window\_size) / window\_size, mode='same')

* **Purpose**: Smooths the data by calculating the average fuel level over a specified window. This method reduces sudden fluctuations in the data.
* **Parameters**:
  + data: The signal data.
  + window\_size: Number of points in the moving average window.

### **4. Connect to MongoDB and Fetch Data**

client = MongoClient("mongodb://localhost:27017/")

db = client['FuelFilter']

collection = db['DeviceGeoHistory']

* **Purpose**: Establishes a connection to MongoDB, selecting the FuelFilter database and DeviceGeoHistory collection where the vehicle's fuel data is stored.

### **5. Set Vehicle and Date Range Filters**

specific\_vehicle\_no = "LL-2501"

start\_date = "2024-09-04"

end\_date = "2024-09-25"

* These variables specify the vehicle (specific\_vehicle\_no) and date range (start\_date to end\_date) for filtering the data.

### **6. Fetch Documents for the Specified Vehicle and Date Range**

documents = collection.find({"vehicleNo": specific\_vehicle\_no})

* **Purpose**: Queries MongoDB to retrieve records of the specific vehicle. Filters could be further added to query data within a specified date range.

### **7. Load Data into a List with Filtering**

data = []

previous\_fuel\_level = None

for doc in documents:

geo\_data = doc.get("geoData", [])

for item in geo\_data:

try:

timestamp = item.get("timeStamp")

if isinstance(timestamp, dict) and "$date" in timestamp and "$numberLong" in timestamp["$date"]:

timestamp = pd.to\_datetime(timestamp["$date"]["$numberLong"], unit='ms')

fuel\_level = int(item["fuelLevelE2"]) if isinstance(item["fuelLevelE2"], dict) else item["fuelLevelE2"]

if fuel\_level == 0 and previous\_fuel\_level is not None:

fuel\_level = previous\_fuel\_level

else:

previous\_fuel\_level = fuel\_level

data.append({

"timestamp": timestamp,

"fuelLevelE2": fuel\_level,

"speed": float(item["speed"])

})

except KeyError as e:

print(f"Missing key in JSON data: {e}")

except Exception as e:

print(f"Error processing item: {e}")

* **Purpose**: Parses each document's geoData, extracting and processing timestamp, fuelLevelE2 (fuel level), and speed. If the fuel level reads 0, the previous non-zero value is used to prevent erroneous readings caused by sensor noise or other anomalies.
* **Error Handling**: Catches missing keys and logs errors.

### 

### **8. Convert Data into a DataFrame and Set Timestamp as Index**

df = pd.DataFrame(data)

df.set\_index("timestamp", inplace=True)

* **Purpose**: Loads the data list into a pandas DataFrame and sets timestamp as the index for time-series analysis.

### **9. Resample Data to a 1-Minute Interval**

resampled\_df = df.resample("60s").mean().ffill()

* **Purpose**: Resamples the data to 1-minute intervals, filling any missing data with the previous available data point (forward fill) for consistent time intervals.

### **10. Filter Application**

cutoff\_frequency = 0.01

sampling\_rate = 3

numtaps = 101

resampled\_df['fuelLevelE2\_smoothed\_m'] = moving\_average\_lowpass\_filter(resampled\_df['fuelLevelE2'], 20)

resampled\_df['fuelLevelE2\_smoothed'] = fir\_lowpass\_filter(resampled\_df['fuelLevelE2\_smoothed\_m'], cutoff\_frequency, sampling\_rate, numtaps)

resampled\_df['smoothed\_fuel\_level'] = resampled\_df['fuelLevelE2\_smoothed']

* **Purpose**: Filters fuelLevelE2 with both moving average and FIR filters to reduce noise.
  + cutoff\_frequency and sampling\_rate control the FIR filter's behavior.
  + fuelLevelE2\_smoothed\_m is the data after the moving average filter.
  + fuelLevelE2\_smoothed applies the FIR filter to further smooth the moving average result.

### 

### **11. Fuel Calibration Using Polynomial Fit**

calibration\_data = pd.DataFrame({

'Actual Liter': [...],

'fuelLevelE2': [...]

})

coefficients = np.polyfit(calibration\_data['fuelLevelE2'], calibration\_data['Actual Liter'], deg=2)

poly\_func = np.poly1d(coefficients)

resampled\_df['fuel\_liters'] = poly\_func(resampled\_df['smoothed\_fuel\_level'])

* **Purpose**: Converts the sensor’s raw fuelLevelE2 values to actual fuel liters using calibration data.
  + calibration\_data holds sample values for fuel level (fuelLevelE2) and corresponding liters (Actual Liter).
  + np.polyfit performs a polynomial fit (degree 2) on this data, creating a polynomial function (poly\_func) that maps fuelLevelE2 to liters.
  + resampled\_df['fuel\_liters'] applies this mapping to the smoothed fuel level values for calibrated fuel level data in liters.

### **Conclusion**

This code handles fuel data collection, noise reduction, smoothing, and fuel level calibration for more accurate real-world measurements. Each critical step — filtering, resampling, and calibration — is necessary to handle sensor inaccuracies and provide meaningful fuel level insights in dynamic vehicle conditions.

Using a Finite Impulse Response (FIR) filter for smoothing fuel sensor data in vehicles is particularly effective in this scenario due to several key factors. Here's a breakdown of why FIR filters are well-suited for handling the challenges of fuel level readings in vehicles:

1. **Stable Filtering with Minimal Phase Distortion**: FIR filters can be designed to have linear phase characteristics, which means that all frequency components of the signal are delayed by the same amount. This linear phase response is crucial in applications like fuel level measurement, where preserving the shape of the signal is important. It ensures that sudden changes in the data, like a spike due to road bumps or inclines, do not distort the true fuel level readings over time.
2. **Reduced Sensitivity to Noise**: Fuel sensors are affected by noise due to various factors, such as road conditions (e.g., bumps, slopes, or turns) and vehicle vibrations. FIR filters are robust to random noise since they are designed to smooth high-frequency fluctuations without affecting the overall trend of the signal. This helps mitigate the noise while retaining the actual fuel consumption pattern, which is vital for accurate readings.
3. **Simplicity and Design Flexibility**: FIR filters are simpler to design and implement compared to Infinite Impulse Response (IIR) filters. For fuel level monitoring, which does not require highly complex filtering, FIR filters offer a straightforward solution with fewer stability concerns. The FIR filter is designed with a specific cutoff frequency that removes high-frequency components, or noise, associated with rapid changes that are unlikely to reflect true fuel level changes.
4. **Memory of Past Data**: Unlike IIR filters, FIR filters only rely on a finite number of past data points. This finite data dependency is advantageous when working with data over short periods, as it makes the filter responsive to new information without being unduly influenced by older data that might contain outliers or irregularities from road disturbances.
5. **Minimized Signal Distortion**: Given that fuel levels typically change slowly over time, a low-pass FIR filter effectively attenuates fast, high-frequency fluctuations that don’t represent actual fuel usage changes. This approach helps maintain the integrity of the data, ensuring that the filtered signal represents a realistic view of fuel levels without introducing unwanted oscillations or distortions that could arise with other filter types.

In summary, an FIR filter effectively balances noise reduction and signal fidelity, making it ideal for smoothing fuel sensor data in vehicles. The filter design parameters (like the cutoff frequency and number of taps) are optimized to preserve the signal’s shape while eliminating high-frequency noise.